BSORT

BSORT is a high speed sort utility for sorting BASIC arrays. Any type of array (integer, single/double precision or string) may be sorted. Sorts can be performed on one and two dimensional arrays. The following syntax is used to initiate a sort. **Note:** "Integer Numbers" refers to integer variables or constants.

```
SYSTEM"RUN BSORT NUM%,*IND%,PSA(x),parm,parm,...,parm"
SYSTEM"RUN BSORT $STRVAR$"

NUM%       Number of elements to sort (an integer number).

*IND%(x)   Optional single dimension integer array. If not
           used, re-ordering of elements will occur in the
           array being sorted. If used, the sort will
           generate an index array containing element
           numbers of the sorted array, and no re-ordering
           of "sorted arrays" will occur.

PSA(x)     Primary sort array. An optional <+> or <-> may
           precede the array name to indicate the direction
           (ascending or descending order) of the sort. If
           not specified, <+> is assumed. A declaration tag
           (!,#,$,%) must be used for any array specified.
           A subscript must be specified, representing the
           first element number to be sorted. It must be an
           integer number.

Optional parameters are as follows:

SSA(x)     Secondary sort array. If used, a <+> or <-> must
           precede the array name. The sort key used will
           include corresponding information from the
           primary and secondary arrays. Any re-ordering of
           the primary array will cause a corresponding
           re-ordering of the secondary array. More than
           one may be used. A subscript is required if the
           secondary array is two dimensional.

TA         Tag array. Any re-ordering in the primary array
           will cause a corresponding re-ordering in a tag
           array. A tag array cannot be preceded by a <+>
           or <->, and may only appear after all secondary
           array definitions. More than one may be used.

(s,n)      Mid-string information. Valid only with STRING
           arrays. If specified, it must immediately follow
           the array information, and cannot be used with
           tag arrays. If specified, the sort key will
           begin at position s in the string, for n
           characters, where s and n are integer numbers.

$STRVAR$   Optional non-array string variable containing
           the sort parameters. Must be used if the length
           of the sort command (i.e. the number of chars.
           within the quote marks) exceeds 79.
```

BSORT can be used to perform many different sorting tasks from within a BASIC program. Only variables and arrays that have been "established" can be used; BSORT cannot allocate memory for variables or arrays. If an un-dimensioned array is used in a sort command, an error will be generated. The following examples illustrate the many different types of sorts which can be performed.

Sorting a Single Dimension Array

Sorting a single dimension array is the simplest type of sort. An integer, single/double precision, or string array may be sorted. In order to sort a one dimension array, two parameters must be passed to BSORT: the number of elements to be sorted, and the starting position in the primary sort array. As an example, assume that the following string array exists in memory:

A$(1)	A$(2)	A$(3)	A$(4)	A$(5)	A$(6)
SMITH	JONES	BROWN	WILLIAMS	JOHNSON	GREEN

To sort the array, this command would be used, with the results shown below.

```
SYSTEM"RUN BSORT 6,+A$(1)"
```

A$(1)	A$(2)	A$(3)	A$(4)	A$(5)	A$(6)
BROWN	GREEN	JOHNSON	JONES	SMITH	WILLIAMS

In the example, the number of elements to be sorted was specified as six, with the sort being performed on array A$ (the primary sort array), starting at element 1.

In this type of sort, a re-ordering of elements takes place. The sequence is in ascending order, so that the value of A$(1)<A$(2)<A$(3) etc. The plus sign <+> in front of the primary sort array indicates that the direction of the sort is to be in ascending order. In this case, the plus sign is optional; if the primary sort array appears without a sign preceding it, the sort will be in ascending order.

Sorting may also be performed so that the re-ordering is in descending order. This is accomplished in the sort command by using a minus sign <-> in front of the primary sort array. Thus, after executing the sort command:

```
SYSTEM"RUN BSORT 6,-A$(1)"
```

the value of A$(1) would be "WILLIAMS"; the value of A$(6) would be "BROWN".

Note that in the above examples, the number of elements to be sorted (6) and the starting array position (1) were specified as integer constants. Any integer constant which needs to be passed to BSORT can be specified as a simple (non-array) variable. The only restriction in using a variable as a value passed to the sort utility is that it must be an integer type, with the type declaration tag (%) explicitly present. DEF statements (e.g. DEF INT) may be used; but the variable as used in the sort command must have a type declaration tag.

Arrays used in BSORT must have a type declaration tag. In the above examples, an error would occur if the commands DEF STR A:DIM A(6) were issued and the following sort invoked, since the <$> declaration tag was not present.

```
SYSTEM"RUN BSORT 6,A(1)"
```

When sorting an array, any part of the array may be sorted for any number of elements. Assuming from the above examples that A$ is dimensioned to have 7 elements {A$(0) through A$(6)}, the following sort can be executed.

```
NM%=4:PO%=2
SYSTEM"RUN BSORT NM%,A$(PO%)"
```

This sequence of commands would cause elements 2 through 5 to be sorted in ascending order, and would leave elements 0,1 and 6 untouched.

An error will be generated if sorting is forced beyond the dimensioned length of the array. In the above example, if PO% is 2, an error will be generated if NM% is assigned a value greater than 5.

Using Secondary Sort Arrays

More than one array may be used to determine the results of a sort operation. Secondary sort arrays can be specified after the primary sort array, and will be included in the sort (i.e. they will aid in determining the direction of the sort and will be re-ordered in conjunction with the primary sort array).

For example, assume that the following arrays are currently active in memory.

A$(1)	A$(2)	A$(3)	A$(4)	A$(5)	A$(6)
SMITH	JONES	JONES	WILLIAMS	JOHNSON	JONES

F$(1)	F$(2)	F$(3)	F$(4)	F$(5)	F$(6)
SAMMY	BILLY	BETTY	RICHARD	CHARLES	BOBBY

The array A$ represents a list of last names, while the array F$ contains the corresponding first name. It is desired to create a sorted list of these names in ascending order, where the first name is used to determine the sorted order when the last names are the same. The following sort command may be used to accomplish this task, with the results shown below.

```
SYSTEM"RUN BSORT 6,A$(1),+F$
```

A$(1)	A$(2)	A$(3)	A$(4)	A$(5)	A$(6)
JOHNSON	JONES	JONES	JONES	SMITH	WILLIAMS

F$(1)	F$(2)	F$(3)	F$(4)	F$(5)	F$(6)
CHARLES	BETTY	BILLY	BOBBY	SAMMY	RICHARD

The array F$ is a secondary sort array. It is used in the sorting process to determine the sorted order when a direct match is found in the primary array. When a secondary sort array is specified, a direct correlation between elements in the primary array is assumed. Any re-ordering which occurs in the primary array will also occur in the secondary array. Thus in this example, the first names were carried along in the sort with the last names, and any exact matches on the last names caused the first names to be sorted.

There are some points that need to be made with respect to syntax and usage of secondary sort arrays. When dealing with a single dimension secondary array, it must be separated from the primary array with a comma. Additionally, no subscript number is required. Any re-ordering which occurs will be done according to the element number in the primary array (i.e. element 1 in the primary array corresponds to element 1 in the secondary array). Futhermore, secondary arrays must be dimensioned as high as the largest sorted element number in the primary array. For example, if a primary array is dimensioned to have 50 elements (0-49) and a secondary array is dimensioned to have 10 elements (0-9), a sort using both arrays could only be performed up to and including element nine. An error will be generated if the sort should go beyond the highest allowable element number of either the primary or secondary array.

Unlike primary arrays, use of a direction sign (+ or -) is mandatory when specifying a secondary array. The direction of the sort in a secondary array does not have to match that of the primary array. Using the above arrays (A$ and F$), the following sort command would produce the results shown below.

```
SYSTEM"RUN BSORT 6,+A$(1),-F$"
```

A$(1)	A$(2)	A$(3)	A$(4)	A$(5)	A$(6)
JOHNSON	JONES	JONES	JONES	SMITH	WILLIAMS

F$(1)	F$(2)	F$(3)	F$(4)	F$(5)	F$(6)
CHARLES	BOBBY	BILLY	BETTY	SAMMY	RICHARD

Note that the directioning of the secondary sort array (in descending order) did not affect the re-ordering of the primary array (ascending order). However, any exact matches in the primary array caused the secondary array (first name array) to be sorted in descending order.

Using Multiple Secondary Arrays

The concept of using more than one secordary array does not differ greatly from using just one secondary array. In terms of syntax, commas must separate subsequent secondary arrays. The same restrictions also apply when more than one secondary array is used (i.e. the mandatory direction sign and the dimensioned lengths of the arrays).

The important point to note is that the order in which the arrays are entered on the sort command line may have an effect on the results of the sort. That is, the first secondary array specified will be the first array used to determine the results of the sort. As an example, examine the three arrays that are shown on the next page.

A$(1)	A$(2)	A$(3)	A$(4)	A$(5)	A$(6)	A$(7)
SMITH	BROWN	JONES	JONES	JONES	JONES	JONES

F$(1)	F$(2)	F$(3)	F$(4)	F$(5)	F$(6)	F$(7)
SAMMY	ROBBY	JOHN	JAKE	JOHN	HERB	HERM

I%(1)	I%(2)	I%(3)	I%(4)	I%(5)	I%(6)	I%(7)
1001	1002	1003	1004	1005	1006	1007

Array A$ contains last names, Array F$ contains first names and Array I% contains ID numbers. Consider the results of the following sort command (shown below).

```
SYSTEM"RUN BSORT 7,A$(1),+F$,-I%"
```

A$(1)	A$(2)	A$(3)	A$(4)	A$(5)	A$(6)	A$(7)
BROWN	JONES	JONES	JONES	JONES	JONES	SMITH

F$(1)	F$(2)	F$(3)	F$(4)	F$(5)	F$(6)	F$(7)
ROBBY	HERB	HERM	JAKE	JOHN	JOHN	SAMMY

I%(1)	I%(2)	I%(3)	I%(4)	I%(5)	I%(6)	I%(7)
1002	1006	1007	1004	1005	1003	1001

The primary sort occurred on the last name (ascending order). If an exact match occurred in the last name, the first name was used to determine the order (ascending order). If two people had identical first and last names, the ID number was then used, and sorted in descending order.

If the secondary arrays were "switched" on the command line, the results obtained would be quite different. Observe this sort command and associated results.

```
SYSTEM"RUN BSORT 7,A$(1),-I%,+F$"
```

A$(1)	A$(2)	A$(3)	A$(4)	A$(5)	A$(6)	A$(7)
BROWN	JONES	JONES	JONES	JONES	JONES	SMITH

F$(1)	F$(2)	F$(3)	F$(4)	F$(5)	F$(6)	F$(7)
ROBBY	HERM	HERB	JOHN	JAKE	JOHN	SAMMY

I%(1)	I%(2)	I%(3)	I%(4)	I%(5)	I%(6)	I%(7)
1002	1007	1006	1005	1004	1003	1001

Note that the last names did sort in ascending order. However, since the I% array appears immediately after the primary array, any identical match found on the last name caused the next sort criteria to be taken from the I% array, with the results being determined in descending order.

Using Tag Arrays

In addition to using secondary arrays, Tag Arrays may be specified on a sort command line. They are similar to Secondary sort arrays, with the exception that the information contained in them has no bearing on the results of the sort. If a tag array is used, any re-ordering which occurs in the primary sort array will also occur in a tag array.

Tag arrays are distinguished from secondary arrays by the lack of a direction sign. If an array (other than the primary array) has no direction sign, it is taken to be a tag array. If both tag and secondary arrays are to be used, ALL secondary arrays must be defined on the sort command line prior to any tag array definitions. Subsequent tag arrays must be separated by commas, and no subscript number is required.

Consider a last name - first name example, where the following arrays have been defined in memory.

A$(1)	A$(2)	A$(3)	A$(4)	A$(5)	A$(6)
JONES	JONES	JONES	WILLIAMS	JOHNSON	JONES

F$(1)	F$(2)	F$(3)	F$(4)	F$(5)	F$(6)
ROBIN	BILLY	BETTY	RICHARD	CHARLES	BOBBY

A typical sort command which makes use of F$ as a tag array could be represented by the following, with the results shown below.

```
SYSTEM"RUN BSORT 6,A$(1),F$"
```

A$(1)	A$(2)	A$(3)	A$(4)	A$(5)	A$(6)
JOHNSON	JONES	JONES	JONES	JONES	WILLIAMS

F$(1)	F$(2)	F$(3)	F$(4)	F$(5)	F$(6)
CHARLES	ROBIN	BILLY	BOBBY	BETTY	RICHARD

Note that the last names are sorted in correct ascending order. However, since F$ was used as a tag array, it did not affect the results of the sort, and only a re-ordering of the data occurred. The order of the items shown in the F$ array is related to the re-ordering of the A$ array. Whenever exact data matches occur (as is the case with the name JONES), re-ordering is done in a random fashion. If additional sort information is required, it should be specified as a secondary sort array. If information is only to be "moved along with" sorted information, it may be specified as a tag array.

MID$ Sorting

When sorting string arrays, an optional MID$ (mid string) parameter may be specified with primary and/or secondary arrays. This will allow sorting to be done on a string array using only a specified part of the string. If re-ordering is performed, the entire string element will be "moved".

As an example, consider the following two string arrays.

L$(0)	L$(1)	L$(2)	L$(3)	L$(4)
D. BROWN	R. SMITH	T. JONES	R. SMITH	P. JONES

F$(0)	F$(1)	F$(2)	F$(3)	F$(4)
BR, DALE	SM, ROB	JO, TERRY	SM, RICH	JO, PETE

Array L$ contains a first name initial, followed by a period, a space and a last name. Array F$ contains the first two characters of the last name, followed by a comma, a space, and the first name. With a sort to be done in the order of last name - first name, this sort command could be used, with the results shown below.

```
SYSTEM"RUN BSORT 5,L$(0)(4,7),+F$(5,6)
```

L$(0)	L$(1)	L$(2)	L$(3)	L$(4)
D. BROWN	P. JONES	T. JONES	R. SMITH	R. SMITH

F$(0)	F$(1)	F$(2)	F$(3)	F$(4)
BR, DALE	JO, PETE	JO, TERRY	SM, RICH	SM, ROB

In this sort command, the primary sort array is the L$ array, while F$ is a secondary array. Both arrays are sorted in ascending order.

The MID$ information immediately follows the subscript position for the primary array. It is enclosed within parentheses, and consists of two integer numbers. The first number specifies the position in the string where the sort criteria begins. Here, the strings in the L$ array are to be sorted starting with the fourth character (i.e. the first character of the last name). The second number tells the sort utility the number of characters to include in the sort from the starting position. Here, seven characters of each string (starting at position four of the string) will comprise the sort key for the primary sort array.

Similarly, MID$ information has been supplied with the secondary sort array. Using the F$ array, the sort key will begin at position 5 (i.e. the first character of the first name) in each element of the array, and extend for 6 characters into each string. Thus, the two arrays are sorted in the order of last name - first name.

Several points need to be made with respect to MID$ sort information. It must always immediately follow the last piece of information associated with the array (i.e. no comma separator is used). When sorting single dimension arrays, this will come after the closing parenthesis of the subscript number for the primary array, and after the declaration tag of the secondary array.

BSORT will NOT check to see if the MID$ values are valid for any string, with the exception that they must not exceed 255. If the starting MID$ position exceeds the entire length of the string in question, a "null" value will be used for that particular element of the array. If the starting MID$ position is within the string, but the number of characters to use for sort criteria is more than what is remaining in the string, only the remaining characters will be used.

For example, A$(1)="HI", A$(2)="BYE" and A$(3)="THIS IS THE END". The following sort commands will produce the results shown below.

Sort Command	Ordering of Elements_
1. SYSTEM"RUN BSORT 3,A$(1)(1,3)"	2,1,3
2. SYSTEM"RUN BSORT 3,A$(1)(2,4)"	3,1,2
3. SYSTEM"RUN BSORT 3,A$(1)(3,2)"	1,2,3

In example 1, the first 3 characters of each string are used in the sort to determine the results. In example 2, the second through fifth characters of each string are used. In example 3, characters three and four of each string are used. Since the first array position only has a length of two characters, its sort value is "null", and so it was sorted "first" (in ascending order).

Generating an Index Array

The previous examples illustrated methods by which arrays were sorted into either ascending or descending order. With those sorts, the array data was re-ordered, so that physical access of the array (by ascending/descending element numbers) was required to see the sorted results. In some cases (such as reading data into an array from a random access file) it may not be desirable to actually re-order an array when "sorting". Thus, BSORT may also be used to generate index arrays. BSORT will initialize the index array to contain the element numbers of the array to sort. The sort will re-order the index array, so that the values in the index array will form a list of pointers to the "sorted" elements of the primary array. For example, assume that the following arrays are currently in memory:

P$(1)	P$(2)	P$(3)	P$(4)	P$(5)	P$(6)	P$(7)
WILLIAMS	SMITH	JONES	BROWN	GREEN	JOHNSON	RICH

I%(1)	I%(2)	I%(3)	I%(4)	I%(5)	I%(6)	I%(7)
0	0	0	0	0	0	0

The sort command listed below could be used to create the index array I%.

SYSTEM"RUN BSORT 7,*I%(1),P$(1)"

P$(1)	P$(2)	P$(3)	P$(4)	P$(5)	P$(6)	P$(7)
WILLIAMS	SMITH	JONES	BROWN	GREEN	JOHNSON	RICH

I%(1)	I%(2)	I%(3)	I%(4)	I%(5)	I%(6)	I%(7)
4	5	6	3	7	2	1

Notice that the sort command did not alter the primary sort array (P$). However, the values in the index array (I%) changed to reflect proper access order of the primary array. Since I%(1) has a value of 4, the fourth element of the P$ array is the first element to access if ascending sorted order is desired. It is now simple to print the P$ array contents in sorted order, using I% as an index.

```
FOR L%=1 TO 7                          FOR L%=1 TO 7
PRINT P$(I%(L%))          or           M%=I%(L%):PRINT P$(M%)
NEXT L%                                NEXT L%
```

When using an index sort, the index array must immediately follow the number of items to sort on the command line, and must be preceded by an asterisk <*> which indicates that an index array is to be generated. The index array MUST be a one dimension integer type with the declaration tag used explicitly. A subscript number must be used with the index array; it will indicate the starting position of the indexed information. This subscript may be either an integer constant or a simple integer variable. Finally, the index array must be dimensioned large enough to contain all index values generated - the number of items sorted. Failure to adhere to these guidelines will more than likely generate an error.

Regarding the subscript number used with the index array, in most cases it will be parallel to the subscript number specified in the sorted array. However, it is not mandatory that these two subscript numbers be the same. As an example, assume that the following integer array exists in memory.

I%(1)	I%(2)	I%(3)	I%(4)	I%(5)	I%(6)	I%(7)	I%(8)	I%(9)	I%(10)
100	200	300	400	500	600	700	800	900	1000

Using this as an index array, the following sort command on the P$ array (see previous example) will produce the results shown.

```
SYSTEM"RUN BSORT 4,*I%(6),P$(2)"
```

I%(1)	I%(2)	I%(3)	I%(4)	I%(5)	I%(6)	I%(7)	I%(8)	I%(9)	I%(10)
100	200	300	400	500	4	5	3	2	1000

This will sort four elements of the P$ array (elements 2 through 5), and store the index information in the I% array, starting at element 6. Note that elements 1-5 and 10 in the index array were unaffected by the sort. The numbers stored in the index array correspond to the element numbers that were sorted (2 through 5).

If the I% array was initially dimensioned to contain 11 elements (0-10), the following sort command would cause an error:

```
SYSTEM"RUN BSORT 4,*I%(8),P$(2)
```

The error would be caused by trying to store index information beyond the end of the index array (i.e. Element #11 of the I% array does not exist).

Indexed sorts may be performed using all of the previously defined sort parameters (i.e. MID$ and secondary arrays). The syntax for such sorting would remain the same, with the exception of the index array being specified in the sort command. No re-ordering will be done on any array used in an indexed sort. Thus, it would be meaningless to use tag arrays in an indexed sort; although no error would be generated, a tag array will not be affected by an indexed sort.

Sorting Two-dimensional Arrays

BSORT supports the use of two dimensional arrays as any type of array (primary, secondary, tag) in a sort command. This section will discuss several variations of using two dimensional arrays.

Throughout the documentation, examples have been given of various sort procedures using single dimension arrays. The array illustrations have alway been depicted in a "horizontal" fashion, representing one row of information with multiple columns. Sorting a one dimension array implies that a row of the array (in this case the only row of the array) be used as the key information, with individual columns being re-ordered (or indexed) to satisfy the requirements of the sort.

This same concept can be carried over to two dimensional arrays. An individual row of the array is specified, from which the key (sort) information is retrieved. Additionally, a starting column number is specified, and the number of elements to be sorted represents the number of columns involved in the sort. If re-ordering is required, an entire column of data is "moved".

As an example, assume that this array (A$) has been established in memory.

		COLUMN				
		1	2	3	4	5
	1	DALE	DAN	DON	DICK	DOCK
R	2	BROWN	JONES	SMITH	GREEN	PETERS
	3	25	34	19	53	42
O	4	BOSTON	BUTTE	BALT	PHIL	PITT
	5	03021	78654	23376	19769	16511
W	6	MA	MT	MD	PA	PA
	7	REP	REP	CLIENT	ADV	STOCK

If it was desired to sort this array by last name in ascending order, the following sort command could be entered, with the results shown below.

```
SYSTEM"RUN BSORT 5,A$(2,1)"
```

		COLUMN				
		1	2	3	4	5
	1	DALE	DICK	DAN	DOCK	DON
R	2	BROWN	GREEN	JONES	PETERS	SMITH
	3	25	53	34	42	19
O	4	BOSTON	PHIL	BUTTE	PITT	BALT
	5	03021	19769	78654	16511	23376
W	6	MA	PA	MT	PA	MD
	7	REP	ADV	REP	STOCK	CLIENT

Several points can be drawn from this example. The total number of items to sort is 5. Row 2 is designated as containing the information to sort. The sort will begin at column 1 (in row 2) and continue for a total of 5 columns. If a re-ordering is to take place, all information in the given column is "moved" (essentially, the two columns involved in the re-ordering are "swapped").

If the A$ array were used as it appeared initially (see Example 1), and the following sort command was issued:

```
SYSTEM"RUN BSORT 2,A$(5,4)"
```

A swap of columns 4 and 5 would be performed. This sort would use information in row 5 as the key. The sort would begin at column 4, and include 2 columns (i.e. columns 4 and 5). Since 16511 is less than 19769, a re-ordering would occur.

Assume once more that the A$ array exists in memory as shown in Example 1. It is desired to generate an index array (I%), where the information in row 3 is sorted in descending order. The following sort command would accomplish this, with the results shown below.

```
SYSTEM"RUN BSORT 5,*I%(1),-A$(3,1)"
```

I%(1)	I%(2)	I%(3)	I%(4)	I%(5)
4	5	2	1	3

Note that when indexing a two dimensional array, the column position of the sorted array is stored in the index array. The sorted array remains unchanged.

Using Two Dimensional Secondary and Tag Arrays

The concept behind sorting two dimensional arrays carries over to the use of two dimensional secondary and tag arrays. In both instances, the number of rows is insignificant. The number of columns in either a secondary or tag array must be as large (or greater than) the highest column number to be sorted in the primary array.

In the case of a tag array, no subscript is required. Re-ordering of columns in the tag array will correspond to those re-ordered in the primary array. The entire column will be "moved", regardless of the number of rows in the array.

The same re-ordering rules apply to two dimensional secondary arrays. However, a subscript must be included with the secondary array. The subscript will be the row number from which key information is to be taken.

Let us assume that the following arrays exist in memory.

A$(1)	A$(2)	A$(3)	A$(4)	A$(5)
BROWN	ADAMS	BROWN	ADAMS	BROWN

ARRAY B$

		COLUMN 1	2	3	4	5
R	1	PRES	VP	MGR	SALES	DIST
O	2	25	53	34	42	19
W	3	DALE	DOCK	DAN	DICK	DON

The A$ array is to be the primary array, and row three of the B$ array will be used as secondary sort information. The following sort command would yield these results (i.e. primary sort by last name, secondary sort by first name), with the sorted arrays being shown on the next page.

```
SYSTEM"RUN BSORT 5,A$(1),+B$(3)
```

A$(1)	A$(2)	A$(3)	A$(4)	A$(5)
ADAMS	ADAMS	BROWN	BROWN	BROWN

B$ ---------- COLUMN ----------

		1	2	3	4	5
R	1	SALES	VP	PRES	MGR	DIST
O	2	42	53	25	34	19
W	3	DICK	DOCK	DALE	DAN	DON

Note that any re-ordering which occurred in the primary array forced the entire (corresponding) column in the B$ array to be moved.

The same re-ordering will occur if a two dimensional tag array is used. A subscript is not required. Entire columns (regardless of the number of rows) will be re-ordered according to the corresponding re-ordering of the primary array.

When using two dimensional secondary arrays, the same array can be used more than once in a sort command, provided that the row specified is different in each case. As a matter of fact, if the primary array is two dimensional, a row different than the primary sort row may be specified as a secondary sort array. In the case of our first example dealing with two dimensional arrays, if it was desired to obtain a sort on this array primarily by last name (row 2) and secondarily by first name (row 1), the following sort command could be used:

```
SYSTEM"RUN BSORT 5,A$(2,1),+A$(1)"
```

This would have the affect of using row 2 of the A$ array as the primary sort information, while using row 1 of the same array as secondary sort information.

As a final point, it is permissible to use the MID$ function when dealing with two dimensional secondary arrays. As is the case with a one dimension primary array, the MID$ information would immediately follow the row subscript of the secondary array. The following example illustrates the syntax that would be used:

```
SYSTEM"RUN BSORT 10,XX%(2,5),+C$(3)(19,8)"
```

Here, XX% is the primary array. The sort will be on row 2 of this array, starting at column 5. It will extend for 10 columns (up to and including column 14). C$ will be used as the secondary array. Columns 5-14 of row 3 will be used. Within each of these elements, a MID$ will be performed, so that the string used in the sort will begin at position 19 of each element, and extend for 8 characters.

Using a Variable to Pass the Sort Command

Depending on the number of parameters specified, the length of an actual sort command may become quite large. A limitation does exist in that the total length of a SYSTEM command cannot exceed 79. For this reason, BSORT allows for the passing of sort parameters in a simple string variable. For example, this command utilizes information contained in PARM$ as the parameters to use for the sort.

```
PARM$ = "10,*II%(1),-AA$(4,1)(15,20),+AC#(3),-SD$(7)(13,8)"
SYSTEM "RUN BSORT $PARM$"
```

In the command which initiates the sort, a <$> must precede the string variable containing the sort parameters. In using this type of sort command, the only limitation is that the length of the string cannot exceed 255 characters.

BSORT/MOD324 - Copyright 1984 MISOSYS., Inc. All rights reserved

MOD324

MOD324 is a utility designed to aid in converting programs created under MODEL III BASIC to MODEL 4 BASIC. The MODEL III program must be contained on a diskette formatted by either MODEL 4 TRSDOS/LS-DOS or MODEL III LDOS (use CONV to move the program from a MODEL III TRSDOS diskette to a MODEL 4 TRSDOS/LS-DOS diskette). The following syntax is used (from the DOS Ready level) to perform a conversion:

```
==============================================================
|  MOD324 filespec1 filespec2 (parm,...,parm)                |
|                                                            |
|  filespec1    MODEL III program to convert. Must be "Saved" |
|               in compressed format. If not specified, it   |
|               will be prompted for.                        |
|                                                            |
|  filespec2    File to contain the converted program. If it |
|               does not exist, the file will be created. If |
|               it exists, the previous contents of the file |
|               will be overwritten. If not specified, it    |
|               will be prompted for. When specified on the  |
|               on the command line, it must appear after   |
|               filespec1.                                   |
|                                                            |
|  Optional parameters are as follows:                       |
|                                                            |
|  MODIFY       Adjust numeric constants in PRINT@ statements |
|               to the corresponding value (absolute row and |
|               column position) on the MODEL 4 video.       |
|                                                            |
|  CENTER=n     Additional offset value which is added to    |
|               all PRINT@ positions changed by MODIFY. Will |
|               work only if MODIFY is specified. Will also  |
|               offset numeric constants in PRINT TAB        |
|               statements according to the column position  |
|               of the value entered. Default is 328.        |
|               (4 lines, 8 columns)                         |
|                                                            |
|  PRINT        Send output of possible manual corrections to |
|               the printer. If not specified, output will be |
|               to video.                                    |
|                                                            |
|  WIDTH=n      Can be used only if PRINT is specified. Will |
|               determine the maximum number of characters to |
|               PRINT per line. Default is 80.               |
|                                                            |
|  abbr:        All parameters may be abbreviated to their   |
|               first character.                             |
==============================================================
```

I M P O R T A N T N O T I C E

MOD324 is designed to be used as an aid in converting programs which are currently running on the MODEL III to a format that can be read by the MODEL 4. Some program commands and sequences which function error free on the MODEL III will NOT work on the MODEL 4. Every attempt is made by MOD324 to flag possible error situations that could result. However, there is NO GUARANTEE (implied or otherwise stated) that a program converted by MOD324 will work, even if no "Manual corrections" were indicated.

BSORT/MOD324 - Copyright 1984 MISOSYS., Inc. All rights reserved

Program Description

MOD324 can be used to convert MODEL III programs to a form that can be read by MODEL 4 BASIC. The MODEL III program must be stored in "compressed format" (i.e. it should NOT have been saved in ASCII). MOD324 will create an ASCII file containing many of the necessary changes to allow the program to be run under MODEL 4 BASIC. Some of the conversions that will take place are:.

1. All "Tokenized" key words and symbols found in the MODEL III program will be changed to the corresponding ASCII representation of the key word/symbol in the MODEL 4 file.

2. Spaces will be inserted into the MODEL 4 text where needed. This includes inserting a space after non-function key words (i.e. those that contain no information within parentheses, such as FOR, TO, NEXT), and after variables/constants which precede a key word, and are not separated from the key word by a terminator (e.g. in the sequence IF A%=10THEN A%=5, a space would be inserted between the <O> of 10 and the <T> of THEN).

3. Any value used in conjunction with a CLEAR statement will be "stripped off". For example, if the statement CLEAR 5000 appeared in the MODEL III program, the resulting statement in the MODEL 4 text would be CLEAR (the function of the CLEAR statement is entirely different on the MODEL 4).

4. Numeric constants used with PRINT@ and PRINT TAB will be adjusted to a corresponding print "position" on the MODEL 4 (if the MODIFY parameter is specified).

There are cases in which no conversions will take place. Any information which appears in the MODEL III program file as ASCII will be left as is. No alterations will be made to either information appearing within quotes, or information following a "Tokenized" REM statement (i.e. the apostrophe character).

Aside from the program conversions that are required, other problems may arise when converting a MODEL III program to run on the MODEL 4. One such source of difficulty is with program statements that exist in MODEL III BASIC but have no meaning on the MODEL 4. Another consideration is in program statements which exist in both BASICs but function differently for one reason or another. Although "translation" of these types of commands would be difficult (if not impossible), MOD324 does provide feedback (i.e. output to the video or printer) on commands that could pose a problem if used with MODEL 4 BASIC.

The following is a list of MODEL III commands that will be "flagged" by MOD324 as possibly needing manual correction.

CLOAD
CMD
CSAVE
ERR
IF (when not followed by THEN)
INP
INPUT #-1 , INPUT #-2
NAME
OUT
PEEK
POKE
POINT
POS
PRINT@
PRINT TAB
PRINT #-1 , PRINT #-2
RESET
SET
SYSTEM
TIME$
USR

PRINT statements (in particular PRINT@ and PRINT TAB) receive special consideration when encountered by MOD324. Although these commands are accepted by MODEL 4 BASIC, video output can cause a problem, since the video sizes differ (64x16 vs. 80x24). For this reason, any occurrence of PRINT@ and PRINT TAB statements will be flagged. There are provisions for MOD324 to adjust values associated with these PRINT statements. Refer to the information on the MODIFY and CENTER parameters for further details.

The last situation which will be flagged by MOD324 is when the resulting conversion would cause a program line to exceed the maximum line length. Due to the "expansion" of key words and the insertion of spaces, a MODEL III program line could be converted into a line which is greater than 254 characters (the maximum line length in MODEL 4 BASIC). When this type of situation occurs, the line will be truncated, and any information in the orginal program line that could not be saved to the MODEL 4 program file would be displayed on the video (or sent to the printer). In this case, a new line will need to be added to the MODEL 4 program, incorporating the "Lost" information. Note: Program logic may be affected by the truncation of a line.

Program Usage

To perform a program conversion, all that is required is to enter <MOD324> at the DOS Ready level. The following prompts to appear (one at a time).

Input Filespec?
Output Filespec?

Pressing <BREAK> in response to either prompt will cause a return to DOS Ready. Any error encountered while answering these prompts (e.g. File not in directory or Write protected disk) will cause the appropriate error message to be displayed, after which the same prompt will re-appear. All entries must follow the rules associated with valid filespecs.

The first prompt is for the name of the MODEL III program. Answer this prompt by entering the associated filespec. If a drivespec is not used, a global search of all active drives will be performed. Please note that if the file has an extension, the extension must be specified (i.e. /BAS is NOT assumed).

The second prompt is for the name of the file which will contain the converted program. If the filespec entered does not exist, it will be created. If the filespec does exist, any information previously contained in the file will be overwritten by the converted program text. It is recommended that a drivespec be included with the output filespec, to assure that the file is written to the proper place. If a drivespec is not entered, the output file will be written to either the "first" drive containing the file, or to the first available drive if the file does not exist on any drive in the system.

Both filespecs may be entered on the command line. For example, if the MODEL 4 program TEST/M4 is to be created (on drive 2) from the MODEL III program TEST/BAS (on drive 1), the following command could be entered.

MOD324 TEST/BAS:1 TEST/M4:2

If only one filespec appears on the command line, it will represent the input filespec, and a prompt will appear for the output filespec.

To see the results of performing a conversion, assume that the following program has been created by MODEL III BASIC, and was saved in compressed form using the

filespec SAMPLE/BAS.

```
10 CLEAR5000:DEFINTA-N:DEFSTRS,T
20 CLS:FORL=1TO10
30 PRINTTAB(5)"This is Line";L;"on  the  MOD III video";TAB(45)"Position
45"
40 NEXT L
```

It is desired to "convert" this program for use on the MODEL 4. The name of the file to contain the converted program is SAMPLE/M4 on drive 2. The following command may be entered to accomplish this.

MOD324 SAMPLE/BAS SAMPLE/M4:2

Two results will occur from the above command. An ASCII file containing the converted program will be created, and feedback for possible manual program corrections (if any) will be given. The first consideration is the program file that is created. The following is a listing of the file SAMPLE/M4.

```
10 CLEAR:DEFINT A-N:DEFSTR S,T
20 CLS:FOR L=1 TO 10
30  PRINT TAB(5)"This is Line";L;"on the MOD III video";TAB(45)"Position
45"
40 NEXT L
```

One point to draw from this listing is the insertion of spaces. Spaces will be inserted as needed. This is clearly illustrated in Lines 10, 20 and 30. Note that in Line 40 no space was added, since one already existed (between the <T> of NEXT and the variable L).

Of additional interest is the resulting CLEAR statement in Line 10. Since the value associated with a MODEL 4 CLEAR statement does not dictate the amount of string space to allocate, any value following a CLEAR statement will be stripped.

In terms of the feedback given (of possible manual corrections), the following information would appear on the video as a result of the conversion performed.

```
The following lines may need manual correction:

30    TAB,TAB

File output completed
```

Any "flagged" key word (see the list on Page 2) that appears in the program will be displayed as the output file is being created. The number of the line containing a flagged key word will be displayed, followed by the key words in question. If multiple key words are flagged on a line, they will be separated by commas. In this example, the key words PRINT TAB appeared twice in Line 30. Note that when TAB appears in a manual correction listing, it is taken to be associated with a PRINT TAB sequence. If TAB is used with an LPRINT statement, no flagging will occur.

After MOD324 has created the output file, it is the sole responsibility of the user to make any manual corrections. In this example, the program could be run as is. However, if any key words were flagged that did not exist in MODEL 4 BASIC (such as SET), they would have to be removed. Furthermore, if key words were found that could cause unpredictable results (such as a POKE of video ram), lines containing these statements would also need to be modified.

PRINT and WIDTH= Parameters

Depending on the length of the program to be converted, the resulting output on manual corrections could become quite lengthy. For this reason, the PRINT parameter has been included. By specifying PRINT, any feedback on possible manual corrections will be sent to the printer (as well as the video).

If PRINT is specified, the WIDTH= parameter may also be used. This will determine the number of characters sent to the printer per line. The default value for WIDTH= is 80. Any value between 9 and 255 may be used.

The printer output will be formatted, so that the line number of a line needing manual corrections will be printed at position 1 (leftmost part) of the line of output. The list of key words will begin at print position 7, and continue for as many key words that exist in the line. If the number of key words to be displayed on the line would cause the WIDTH to be exceeded, the line will be broken at the key word preceding the one causing the "wrap around" (if possible). All remaining key words will then be printed on the next line, starting at print position 7.

Assume it is desired to obtain printed output of possible manual corrections when converting the program SAMPLE/BAS to SAMPLE/M4. The total length of an output line is not to exceed 60 characters. The following command will accomplish this.

```
MOD324 SAMPLE/BAS SAMPLE/M4 (P,W=60
```

MODIFY and CENTER= Parameters

A definite problem can arise with respect to "screen formatting" when converting a MODEL III program to run on the MODEL 4. Consider the following (MODEL III) program, which draws a box on the first 15 lines of the video, prints an informative message on the last line, and blinks a message in the middle of the box.

```
5 CLEAR 2000
10 CLS
20 PRINT@0,CHR$(151);STRING$(62,131);CHR$(171)
30 PRINT@64,CHR$(149):PRINT@127,CHR$(170)
40 PRINT@128,CHR$(149):PRINT@191,CHR$(170)
50 PRINT@192,CHR$(149):PRINT@255,CHR$(170)
60 PRINT@256,CHR$(149):PRINT@319,CHR$(170)
70 PRINT@320,CHR$(149):PRINT@383,CHR$(170)
80 PRINT@384,CHR$(149):PRINT@447,CHR$(170)
90 PRINT@448,CHR$(149):PRINT@511,CHR$(170)
100 PRINT@512,CHR$(149):PRINT@575,CHR$(170)
110 PRINT@576,CHR$(149):PRINT@639,CHR$(170)
120 PRINT@640,CHR$(149):PRINT@703,CHR$(170)
130 PRINT@704,CHR$(149):PRINT@767,CHR$(170)
140 PRINT@768,CHR$(149):PRINT@831,CHR$(170)
150 PRINT@832,CHR$(149):PRINT@895,CHR$(170)
170 PRINT@896,CHR$(181);STRING$(62,176);CHR$(186);
175 PRINT@960,"";TAB(15)"Press Any Key to end this Program";
180 PRINT@473,"Center of Box";
190 I$=INKEY$:IFI$<>""THENEND
200 FORL=1TO30:NEXTL
210 PRINT@473,"              ";
220 I$=INKEY$:IFI$<>""THENEND
230 FORL=1TO20:NEXTL:GOTO180
```

Assuming that this program has been saved as CENTER/BAS, the following conversion

command will produce the feedback output shown.

```
MOD324 CENTER/BAS CENTER/M4:3

File CENTER/M4:3

The following lines may need manual correction:

20    PRINT@(0)
30    PRINT@(64),PRINT@(127)
40    PRINT@(128),PRINT@(191)
                .
                .
                .
150   PRINT@(832),PRINT@(895)
170   PRINT@(896)
175   PRINT@(960),TAB
180   PRINT@(473)
210   PRINT@(473)
```

In this example, all PRINT@ commands use numeric constants to represent print positions. The converted program (CENTER/M4) could be run without performing manual corrections. However, the results would not produce a box on the screen.

In situations similar to this one, the MODIFY parameter may be used. MODIFY will adjust PRINT@ positions which are represented by numeric constants. The output program file will contain these adjusted values, and the feedback will show both the orginal and adjusted values. The original PRINT@ position will be divided by 64 to obtain an integer quotient and remainder. These numbers correspond to the row and column of the PRINT@ position, offset from 0. The adjusted PRINT@ value is obtained by multiplying the row value by 80 and adding in the column number.

The following command will perform a conversion of the program CENTER/BAS, incorporating the MODIFY parameter. The feedback output is shown below.

```
MOD324 CENTER/BAS CENTER/M4:3 (M)

File CENTER/M4:3

The following lines may need manual correction:

20    PRINT@(0=>0)
30    PRINT@(64=>80),PRINT@(127=>143)
40    PRINT@(128=>160),PRINT@(191=>223)
50    PRINT@(192=>240),PRINT@(255=>303)
60    PRINT@(256=>320),PRINT@(319=>383)
70    PRINT@(320=>400),PRINT@(383=>463)
80    PRINT@(384=>480),PRINT@(447=>543)
90    PRINT@(448=>560),PRINT@(511=>623)
100   PRINT@(512=>640),PRINT@(575=>703)
110   PRINT@(576=>720),PRINT@(639=>783)
120   PRINT@(640=>800),PRINT@(703=>863)
130   PRINT@(704=>880),PRINT@(767=>943)
140   PRINT@(768=>960),PRINT@(831=>1023)
150   PRINT@(832=>1040),PRINT@(895=>1103)
170   PRINT@(896=>1120)
175   PRINT@(960=>1200),TAB
180   PRINT@(473=>585)
210   PRINT@(473=>585)
```

In examining the adjustments made to Line 40, the original PRINT@ position of 191 was translated into 223 (row 2, column 63). Running the program CENTER/M4 would cause a box to be drawn on the upper left hand corner of the screen. Manual correction of the program would not be required. Notice that PRINT TAB commands (see Line 175) are not adjusted in the case of a MODIFY, as they refer to column position only.

Because the MODEL 4 video is larger than that of the MODEL III, it is possible to "overlay" a MODEL III screen onto a portion of the MODEL 4 video. The amount of screen movement available is up to 8 rows, 16 columns. In terms of performing a program conversion, the CENTER= parameter may be used in conjunction with the MODIFY parameter, to further adjust PRINT@ positions represented by numeric constants. The default value for the CENTER= parameter is 328 (4 rows, 8 columns).

The following command will perform a conversion of the program CENTER/BAS so that the "box" will be drawn on the center of the MODEL 4 screen. The resulting feedback output is shown below.

MOD324 CENTER/BAS CENTER1/M4:3 (M,C)

```
File CENTER1/M4:3

The following lines may need manual correction:

20    PRINT@(0=>328)
30    PRINT@(64=>408),PRINT@(127=>471)
40    PRINT@(128=>488),PRINT@(191=>551)
50    PRINT@(192=>568),PRINT@(255=>631)
60    PRINT@(256=>648),PRINT@(319=>711)
70    PRINT@(320=>728),PRINT@(383=>791)
80    PRINT@(384=>808),PRINT@(447=>871)
90    PRINT@(448=>888),PRINT@(511=>951)
100   PRINT@(512=>968),PRINT@(575=>1031)
110   PRINT@(576=>1048),PRINT@(639=>1111)
120   PRINT@(640=>1128),PRINT@(703=>1191)
130   PRINT@(704=>1208),PRINT@(767=>1271)
140   PRINT@(768=>1288),PRINT@(831=>1351)
150   PRINT@(832=>1368),PRINT@(895=>1431)
170   PRINT@(896=>1448)
175   PRINT@(960=>1528),TAB(15=>23)
180   PRINT@(473=>913)
210   PRINT@(473=>913)
```

In examining Line 40, the original PRINT@ position of 191 was translated into 551. The MODIFY value of 223 was first obtained. Then, the CENTER value of 328 was added in, to obtain the final result. Running the program CENTER1/M4 would cause a box to be drawn in the center of the screen (the upper left corner of the box is positioned at row 4, column 8). Manual correction of the program would not be required. Notice that PRINT TAB commands (see Line 175) are adjusted in the case of a CENTER, as movement of the entire screen affects column positioning. The value that will be added to numeric constants in PRINT TAB statements is the column offset (in this example, 8). If zero is used as a column offset (i.e. if CENTER=80, 160, 240, etc.), PRINT TABs will not be adjusted by CENTER.

When using the CENTER= parameter, the MODIFY parameter must also be specified for any adjustments to occur. Although any value may be used with CENTER=, some values (e.g. CENTER=99) will produce undesireable results. Offsets of more than 8 rows and/or 16 columns should be avoided. The following table lists the CENTER= value ranges that make the most practical sense.

CENTER= Range	Row Offset
0-16	0
80-96	1
160-176	2
240-256	3
320-336	4
400-416	5
480-496	6
560-576	7
640-656	8

Miscellaneous "Feedback" Information

When PRINT@ and PRINT TAB statements utilize numeric expressions as print position values, adjustments to the positioning values will not be made. However, the feedback associated with such commands will indicate that the print positioning value is a numeric expression. Consider the following MODEL III program (CNTLOOP/BAS) which will draw a box on the video via a FOR-NEXT loop.

```
5 CLEAR 2000
10 CLS
20 PRINT@0,CHR$(151);STRING$(62,131);CHR$(171)
25 FORL=1TO13:A1=L*64:PRINT@A1,CHR$(149):PRINT@A1+63,CHR$(170):NEXTL
170 PRINT@896,CHR$(181);STRING$(62,176);CHR$(186);
172 MC$="Center of Box":MB$="Press Any Key to end this Program"
174 M1=LEN(MC$):M2=LEN(MB$):CM=7*64+((64-M1)/2)
175 PRINT@960,"";TAB((64-M2)/2);MB$;
180 PRINT@CM,MC$;
190 I$=INKEY$:IFI$<>""THENEND
200 FORL=1TO30:NEXTL
210 PRINT@CM,STRING$(M1,32);
220 I$=INKEY$:IFI$<>""THENEND
230 FORL=1TO20:NEXTL:GOTO180
```

The following command can be used to convert this program, with the resulting feedback output shown below.

MOD324 CNTLOOP/BAS CNTLOOP/M4:3 (M,C)

```
File CNTLOOP/M4:3

The following lines may need manual correction:

20    PRINT@(0=>328)
25    PRINT@(EXP),PRINT@(EXP)
170   PRINT@(896=>1448)
175   PRINT@(960=>1528),TAB(EXP)
180   PRINT@(EXP)
210   PRINT@(EXP)
```

Notice that an adjustment did occur in Line 20. However, in Line 25 the print position was specified as a numeric expression. In this case, an adjustment is not made to Line 25 in the output filespec (converted program). Rather, the feedback message associated with the PRINT@ statement indicates that an expression (EXP) follows the PRINT@. PRINT@(EXP) will always be displayed (regardless of the conversion parameters specified) when a numeric expression follows a PRINT@ statement.

The same type of feedback will occur with PRINT TAB statements. This will happen when a column offset is dictated by the CENTER parameter, and a numeric expression denotes the tab position (see Line 175).

Due to the expansion that takes place during a program conversion (e.g. spaces being inserted), it may be necessary for MOD324 to trucate a program line. Line truncation is done so that the resulting program file may be loaded into memory by MODEL 4 BASIC. When a line is truncated, as much of the line as possible is stored in the output program file, and a feedback message shows the part of the line that was truncated.

As an example, assume that the following line exists in a MODEL III program file.

```
10 FORLL=1TO10:FORLK=1TO20:FORLP=1TO30:LPRINTTAB(20)"This is an example of a
   converted line being too long":LPRINTTAB(20)"The value of LL is";LL:
   LPRINTTAB(20)"The value of lk is";LK:LPRINTTAB(20)"The value of LP is";
   LP:NEXTLP:NEXTLK:NEXTLL:PRINTTAB(20)"Done"
```

Consider the results of performing a conversion of this line, as shown below (shown first is the line as it would be saved to the output filespec, followed by the feedback message that would be generated).

```
10 FOR LL=1 TO 10:FOR LK=1 TO 20:FOR LP=1 TO 30:LPRINT TAB(20)"This is an
   example of a converted line being too long":LPRINT TAB(20)"The value of
   LL is";LL:LPRINT TAB(20)"The value of lk is";LK:LPRINT TAB(20)"The value
   of LP is";LP:NEXT LP:NEXT LK:NEX
```

```
The following lines may need manual correction:

10    TAB
10     - Line truncated, should be extended as follows:
T LL:PRINT TAB(20)"Done"
```

Of interest in this example is the ending part of the line in the output file and the information in the "Line Truncated" feedback meesage. Note that any part of the program line that could not get written to the output file is displayed in the feedback message.

One last point which needs to be mentioned concerns the use of IF-THEN statements. In a MODEL III program, the following type of statement is allowable, and would function without error.

```
IF A=1 A=2
```

In this case, THEN is implied. However, using this type of implied THEN statement on the MODEL 4 would generate a syntax error. For this reason, MOD324 will flag any IF statement which is not followed by a THEN.